Leveraging Linked Data Fragments for enhanced data publication: the Share-VDE case study

Andrea Gazzarini[1,2]

[1]SpazioCodice SRL, 01039 Vignanello (VT), Italy
[2]@Cult/Casalini Libri, 50014 Fiesole (FI), Italy

Abstract

In big data-driven environments, accessing, querying, and processing vast datasets efficiently is challenging.

Linked Data Fragments (LDF) have emerged as a promising paradigm to address these challenges by providing a distributed and scalable approach for publishing and serving Linked Data.

As part of the Share-VDE initiative, we developed a set of Web APIs that adopt Linked Data Fragments and provide the following benefits: real-time RDF generation and publication, on-demand ontology mapping, and multi-provenance management.

Keywords

Resource Description Framework (RDF), Linked Data Fragments (LDF), Triple Pattern, querying, scalability, BIBFRAME, Web API, SPARQL, Share-VDE,

1. Introduction

Libraries, archives, and museums hold extensive data collections. Unveiling the potential to harness and disseminate such information to a broader audience could enhance the expansion of the World Wide Web, foster a culture of knowledge openness, and yield numerous benefits for all stakeholders in the information ecosystem.

Share-VDE[1] is a library-driven initiative to establish procedures for identifying and reconciling entities, converting data to linked data, and creating a virtual discovery environment based upon the three-layered structure of the BIBFRAME[1] data model.

We aim to showcase our implementation efforts in enabling real-time, multi-provenance, and multi-mapping RDF publication by introducing an RDF API layer built upon the innovative concept of Linked Data Fragments[2].

2. Entities as “Prisms”

Share-VDE manages a multi-tenant Knowledge Base comprising clustered, integrated, and enriched entities. A tenant is a set of institutions contributing to the same Knowledge base.

DCMI-2024 International Conference on Dublin Core and Metadata Applications
a.gazzarini@spaziocodice.com (A. Gazzarini)
https://www.linkedin.com/in/andreagazzarini/ (A. Gazzarini)
0000-0002-4567-8901 (A. Gazzarini)

[1]https://svde.org

https://doi.org/10.23106/dcmi.952407512

Data flows from each institution, and it is processed to create the entities that form the Knowledge Base.

An institution within a tenant is called a Provenance. The term comes from the PROV-O[3] ontology, and it is a crucial concept within the Share-VDE dataset: each entity definition is contributed by multiple institutions, and its properties retain the ownership metadata so the system always knows who contributed what.

In the example below, a library produces a bibliographic description representing a work, and then it is sent to Share-VDE. Assuming the system sees that work for the first time, it creates a new entity (the big triangle in the picture) and assigns it a new Uniform Resource Identifier (URI).[2]. The triangle contains the properties derived from the input record; each property is marked as having been contributed by the library. Note the smaller triangles beside the big one, which indicate other entities that could potentially be created.[3] When other libraries send their data, the system creates the additional faces using their contributions: the union of all faces belonging to the same entity forms a prism.

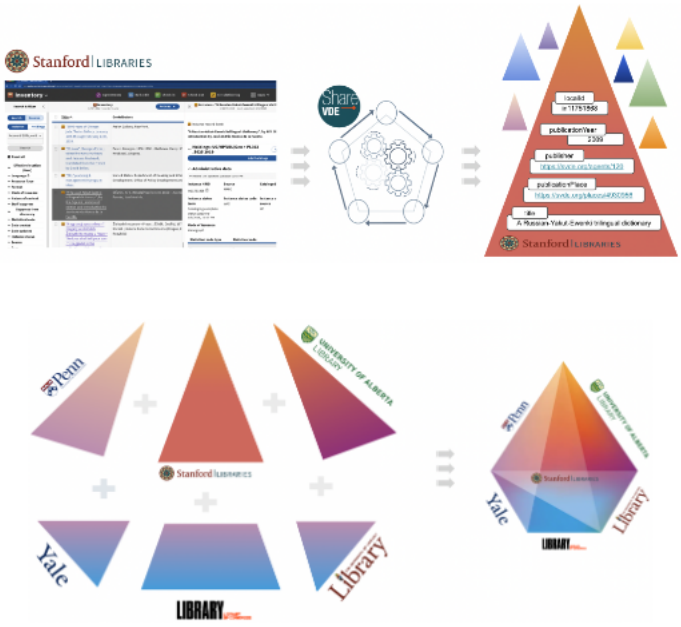

Figure 1: The prism genesis.

Therefore, each entity in the Share-VDE Knowledge Base is a Prism:

- providing a uniform and integrated view of a given concept (e.g., a Work, an Agent, a Place)
- retaining, on each face, the data contributed by a given library
- assigned to a unique, dereferenceable URI.

[2]For example, https://svde.org/publications/26211656926171125

[3]A bibliographic description includes other resources, such as places and agents, other than the work itself.

https://doi.org/10.23106/dcmi.952407512

3. Linked Data Fragments

RDF Data can be provided as a service[4]: an HTTP endpoint where clients can request data through a structured language like SPARQL[5].

A SPARQL endpoint offers clients powerful access to data. Yet, when the dataset grows, it comes with significant infrastructure costs: reliable and scalable commercial products are usually expensive[4]. How can we provide RDF data and, at the same time, reduce server resource consumption while maintaining efficient data querying?

Linked Data Fragments represents an elegant approach that helps answer the question above. A Linked Data Fragments architecture[6] divides the query execution and computation responsibilities among two players: a *Linked Data Fragment Client*[7] that provides the SPARQL endpoint, the query pre-processing, its optimization, and the destructuring of such a query into a set of atomic clauses that can be executed independently towards distributed *Linked Data Fragment Servers*, the second role.

See the following SPARQL query that traverses the Share-VDE core hierarchy:

```
SELECT ?itemUri
WHERE {
    opuses:401 bf:hasExpression ?workUri .
    ?workUri bf:hasInstance ?instanceUri .
    ?instanceUri bf:hasItem ?itemUri .
}
```

Listing 1: Example SPARQL

The query can be destructured in three separated clauses corresponding to three *triple patterns*. Each clause selects a *fragment* of the entire result; the idea is to expose a service that resolve the selector pattern, and returns back the corresponding (linked) data fragment. The compounding fragments of a query response are then merged for creating the final result that is sent back to the client.

A Linked Data Fragment consists of a selector, metadata (variables, counts, page size), and controls (hypermedia links or URIs to other fragments). Examples of selectors are

- give me the first 50 triples whose subject is a given URI
- give me the second page of the triples whose subject is a given URI and predicate is foaf:name
- give me the triples whose object is the literal "Andrea"@it

A triple pattern server[8], that is, a service that answers to a triple pattern request, serves many small/straightforward requests. As a consequence, the execution requires a low/medium computation effort compared to a central server that controls the whole query execution.

Everything comes with a price: the Linked Data Fragment Client can perform minimal optimizations[9] compared with a centralized engine that executes the whole query; also, the distributed nature of the read path[7] necessarily introduces a higher network latency.

[4] https://www.w3.org/wiki/LargeTripleStores

https://doi.org/10.23106/dcmi.952407512

4. On-demand RDF Publication: the RDFizer

What is behind a system that provides RDF data? Generally speaking, there is dedicated storage, usually referred to as a *Triple Store*, *Quad Store*, or the more general name, *RDF Store*.

As the name suggests, a triple store manages RDF natively; it also exposes a SPARQL endpoint[4] for querying the data. While it is definitely powerful, it assumes a system uses it as underlying datasource. Share-VDE uses an RDBMS as a primary storage: data is stored using an agnostic, RDF-unaware shape. How can such a system provide RDF data through a SPARQL endpoint without having a dedicated RDF storage?

The adoption of the Linked Data Fragment architecture[2] makes possible to move the RDF publication and generation entirely on the *query-time* side, therefore enabling the capability of translating data in RDF in real time. The Share-VDE Linked Data Fragment Server receives a triple/quad pattern request, reads the data in the RDBMS, according to a given mapping translates the resultset into RDF, and then it returns the corresponding fragment to a Linked Data Fragment client, that assembles that together with the other response fragments. The response is then sent back to the client.

Besides providing RDF without having dedicated RDF storage, such an architecture enables interesting features, as described in the following paragraphs.

4.1. Multi-mapping

In Share-VDE, data is stored in a relational database using an agnostic format. That means it is not natively available in RDF[5]: it is translated on demand by applying a conversion mapping. Mappings are included in the requests, at query-time: for example, one client would be interested in receiving data in RDF using a given mix of ontologies (e.g., BIBFRAME[6] and RDA[7]) while another client could want a different mix (e.g., FRBROO[8] and Dublin Core[9]). Mappings and mixes are associated with a mnemonic code; clients can use a dedicated HTTP header[10] to indicate the shape of the data they would like to get back in response at query time.

```
GET /sparql?query=CONSTRUCT { ?s ?p ?o } WHERE { ?s ?p ?o }
Host: svde.org
Accept: text/turtle, application/rdf+xml
svde-mapping: bibframe+rda

@prefix opus: <https://svde.org/opuses/>
@prefix svde: <https://svde.org/properties/>
@prefix rdau: <http://rdaregistry.info/Elements/u/>
```

[5]The sentence refers to the new RDFizer architecture described in this document. Instead, the current version uses a plain RDF Store to serve RDF data.
[6]https://www.loc.gov/bibframe/
[7]https://www.rdaregistry.info/
[8]https://www.cidoc-crm.org/frbroo/home-0
[9]https://www.dublincore.org/specifications/dublin-core/
[10]https://datatracker.ietf.org/doc/html/rfc7230#section-3.2

https://doi.org/10.23106/dcmi.952407512

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix bf: <http://id.loc.gov/ontologies/bibframe/>.

...

opus:401  bf:title       svde:123  .
svde:123  rdf:type       bf:Title  .
svde:123  bf:mainTitle   "Proin molestie" .
svde:123  bf:subTitle    "ipsum" .
opus:401  rdau:P60355    "Proin molestie: ipsum" .

...
```

Listing 2: Example SPARQL exchange using a BIBFRAME + RDA mapping

```
GET /sparql HTTP/1.1
Host: svde.org
svde-mapping: dc

@prefix opus:     <https://svde.org/opuses/>
@prefix dcterms:  <http://purl.org/dc/terms/>
@prefix rdf:      <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.

...

opus:401 dcterms:title        "Proin molestie: ipsum" .
opus:401 dcterms:alternative  "Molestie Proin" .

...
```

Listing 3: Example SPARQL query exchange using a DC mapping

4.2. Prism face(s) selection

Share-VDE entities are prisms whose faces represent the different contributions of the consortium participants. A client can indicate at request time, through a dedicated HTTP header[11], one or more institutions. In that case, the query execution operates only over the contributions of the indicated provenances. Instead, if that header is missing, the results would consider all the prisms without applying any face selection.

[11] https://datatracker.ietf.org/doc/html/rfc7230#section-3.2

https://doi.org/10.23106/dcmi.952407512

Listing 4: SPARQL query with a provenance selector header.

```
GET /sparql HTTP/1.1
Host: svde.org
svde-provenances:
    https://svde.org/agents/stanford,
    https://svde.org/agents/yale

... (triples belonging to Stanford and Yale contributions) ...
```

5. Conclusion and future works

The query-time flexibility introduced by Linked Data Fragments paradigm looks promising from a modularity and extensibility perspective. The new infrastructure opens the Share-VDE RDF layer to new challenges: additional mappings, more articulated fragment selector expressions, and write paths (the current implementation focuses only on the read path).

References

[1] Library of Congress, Bibliographic Framework Initiative (BIBFRAME), 2016. URL: https://www.loc.gov/bibframe/.

[2] R. Verborgh, M. V. Sande, P. Colpaert, S. Coppens, E. Mannens, R. V. de Walle, Web-Scale Querying through Linked Data Fragments, 2014. URL: https://ceur-ws.org/Vol-1184/ldow2014_paper_04.pdf.

[3] "W3C", PROV-O: The PROV Ontology, 2013. URL: https://www.w3.org/TR/prov-o/.

[4] W3C, SPARQL 1.1 Protocol, 2013. URL: https://www.w3.org/TR/sparql11-protocol/.

[5] W3C, SPARQL Query Language for RDF, 2008. URL: https://www.w3.org/TR/rdf-sparql-query/.

[6] R. Verborgh, O. Hartig, B. D. Meester, G. Haesendonck, L. D. Vocht, M. V. Sande, R. Cyganiak, P. Colpaert, E. Mannens, R. V. de Walle, Querying Datasets on the Web with High Availability, 2014. URL: https://linkeddatafragments.org/publications/iswc2014.pdf.

[7] J. V. Herwegen, R. Verborgh, E. Mannens, R. V. de Walle, Query Execution Optimization for Clients of Triple Pattern Fragments, 2016. URL: https://linkeddatafragments.org/publications/eswc2015.pdf.

[8] R. Taelman, R. Verborgh, E. Mannens, Exposing rdf Archives using Triple Pattern Fragments, 2016. URL: https://www.rubensworks.net/raw/publications/2016/ExposingRdfArchivesUsingTpf.pdf.

[9] R. Taelman, J. V. Herwegen, M. V. Sande, R. Verborgh, Optimizing Approximate Membership Metadata in Triple Pattern Fragments, 2020. URL: https://comunica.github.io/Article-SSWS2020-AMF/.

https://doi.org/10.23106/dcmi.952407512